MANUEL AHIJADO *

Computación en programación no-lineal y algorítmos de punto fijo

«Una vez asesinado el imposible, será fácil trabajo lo que ser no pueda».
W. Shakespeare. Coriolano. p. 25.

Esta nota tiene un doble objetivo; de un lado presentar dos algoritmos de computación en programación no-lineal, y de otro indicar o mostrar la íntima relación entre ellos y los (correspondientes) de computación de puntos fijos.

Consta de tres secciones. En la primera se expone un skecht de la relación entre ambos tipos, así como el tipo de supuestos genéricos exigidos por los algoritmos; en la segunda y tercera, respectivamente se describen brevemente las características y las formas de operar de test debidos a Colville (1968).

I

Consideramos el siguiente problema supongamos que $(g_t;\ t \in T)$ es un conjunto de funciones numéricas definidas sobre R^n, donde T es un conjunto compacto indicado. Supongamos también que O es una función numérica definida sobre R^n, y sea h la relación definida por

(a) $h : R^n \to R^1$
(b) $h(x) = \sup\ (g_t(x);\ t \in T)$, y
(c) $T(x) = (t \in T;\ g_t(x) = h(x))$.

A esta h así definida se le denomina *función sup.* Si T(x) es un subconjunto vacío de T para cada $x \in R^n$, se dice que h es una *función max.*

Universidad Complutense. Madrid.
* Estoy muy agradecido al profesor Julio Segura por la lectura y comentarios a dos borradores previos. Cualquier error que pueda subsistir es exclusivamente mío.
Esta nota corresponde a la parte instrumental de un trabajo de mayor envergadura, y es un resultado tangencial de las prácticas del autor en lenguaje APL.SV., sobre el sistema IBM 360-65, y en un terminal 2740 del Centro de Cálculo de la Universidad Complutense de Madrid.

El problema consiste en minimizar una función max. Este problema se puede reformular como un problema de punto fijo.

Se trata de hallar un $x' \in R^n$ tal que:

(i) $h(x') \leqq o$, y
(ii) $O(x') = \inf(o\ O(x);\ x \in \text{lev}_o\, h)$,

o determinar en su caso que no existe un x' tal, donde son aplicables las siguientes definiciones debidas a Rookafeller (1970):

(1) Sea $\text{lev}_a O = (x;\ O(x)) \leqq a$, para todo $a \in R^1$. Si $B = \inf(O(x);\ x \in R^n)$, entonces $\text{lev}_B O$ se dice que es el *conjunto mínimo de O.*

(2) Supongamos que C es un conjunto convexo no-vacío en R^n. $y \in R^n$ se dice que es una *dirección de recesión de C*, si y sólo si $x + \lambda y \in C$, para todo $\lambda = o$. El conjunto de todas las direcciones de recesión, incluyendo a o se denomina *cono de recesión de C*, y se denota por $O^+ C$. y se dice que es una *dirección de recesión de una función convexa O*, si y sólo si, la dirección de $\text{lev}_a O$ para algún $a \in R^1$.

A O se le denomina función objetivo, y a g'_t funciones restricción. El problema normalmente se escribe

$$\min O(x)$$
$$\text{S.T. } g(t) \leqq o \quad t \in T$$

Cuando g'_t y O satisfacen ciertas condiciones, el problema anterior se puede formular como un problema de punto fijo.[1]

Cuando tanto g'_t como O son funciones convexas, la convergencia algorítmica (i.e., mediante el segundo algoritmo de la segunda sección) se demuestra ocurre siempre que O y h no tengan una dirección de recesión común distinta de o.

Cuando la función objetiva y/o las restricciones no son convexas, no existen condiciones que garanticen la convergencia algorítmica deseada, o incluso si converge a un punto, éste no necesariamente es la solución del problema; por último, bajo condiciones muy fuertes (i.e., diferenciabilidad continua) se pueden obtener óptimos locales mediante la sustitución del gradiente por un subgradiente. Para repetir, las aplicaciones consideradas «padecen» de dificultades computacionales en el sentido de que las aplicaciones satisfacen condiciones necesarias pero no suficientes de óptimo (global).

Minimización condicionada de funciones convexas

Supondremos que se mantienen los siguientes supuestos:

1. Se puede encontrar un estudio detallado en Merrill (1971).

Supuesto 1. O y (g_t; $t \in T$) son funciones convexas, cerradas, con dom $g_t = R^n$, para todo $t \in T$.
Supuesto 2. lev$_0$h C int (dom O), donde h es la función max definida antes.

Comenzamos desarrollando una formulación de punto fijo del problema mediante

$$F(x) = \begin{cases} x - O(x) & \text{si } h(x) < 0 \\ x - \text{conv}\,(O(x) & h(x) \text{ si } h(x) = 0 \\ x - h(x) & \text{si } h(x) > 0 \end{cases}$$

Es necesario el siguiente teorema auxiliar:

Teorema 1. Supongamos que se mantienen los supuestos 1 y 2, $F : R^n \to p_n$ definida para el problema, es una aplicación punto-conjunto semicontínua superiormente.

Los tres siguientes teoremas[2] establecen las condiciones suficientes que garantizan que los algoritmos de computación de puntos fijos de la segunda sección convergen a un punto fijo de F. También describen la relación entre un punto fijo de F, y una solución al problema de programación planteado.

Teorema 2. Supongamos adicionalmente a la satisfacción de los supuestos 1 y 2, que las funciones de (g_t; $t \in T$) no tienen una dirección de recesión común distinta de cero. Entonces el algorítmo de Merrill (por ejemplo, véase sección II) siempre converge a un punto fijo de la aplicación F.[3]

Teorema 3. Supongamos que se dan las condiciones del teorema 2, y supongamos que el algorítmo mencionado se utiliza para computar un punto fijo de F. Sea $a = \inf (h(x); x \in R^n)$. Entonces:

a. Si $a > 0$, entonces esta condición se puede hallar en un número finito de pasos.
b. Si $a = 0$, entonces cualquier punto fijo de F es factible (i.e., todo punto fijo de F está sobre lev$_0$h.
c. Si $a < 0$, entonces $x' \in F(x')$ es una condición necesaria y suficiente para que x' minimice O sobre lev$_0$h.

También, y por tanto existe al menos un punto fijo.

Teorema 4. Supongamos que se mantienen los supuestos 1 y 2, h y O no tienen direcciones de recesión distintas de o; existe un $x \in R^n$ tal

2. Todos ellos se deben a Merrill (1972).
3. El argumento se extiende fácilmente al algorítmo de Eaves-Saigal. Véase Todd (1975).

que h (x) < o. Sea $\mu > o$, dado. Si para algún x' lev$_o$h, B (x', μ) C int (lev$_o$h), entonces existe un $M < \infty$ tal que se cumplen las hipótesis que demuestran el teorema del algoritmo de Merrill, es decir, existen M, μ, y x', cuando F viene definida por la F del problema.

II

Sea C un subconjunto de R^n, y f : C → C^x, donde C^x es la colección de todos los subconjuntos de C. Un punto de $x \in C$ se dice a que es un punto fijo si y sólo si, $x \in$ f (x).

Ilustraremos dos clases de algorítmos de tercera generación. Frente a los de primera y segunda generación los aquí descritos poseen dos ventajas computacionales importantes:

1. Refinan automáticamente la retícula[4] (i.e, todos ellos tienen en común que operan sobre simplex o espacios euclídeos denominados retículas) a medida que el algoritmo progresa.
2. Parten de una estimación inicial cercana a la región de interés.

Actuan por pivotación, y su convergencia se basa en el método de la senda complementaria de Lemke y Hobson (1964) lo que asegura que no reciclan. La finitud es una característica de cualquier aplicación concreta, y al igual que en la programación lineal, se explota la idea de que el equilibrio se obtiene en un número muy razonable de iteraciones.

El algoritmo de Eaves-Saigal (o algoritmo de la clase R^n x (o, ∞))[5]

Es el único miembro de la clase, y es una extensión del algoritmo de Eaves (1972) sobre S^n x (o, ∞), que utiliza funciones f : $R^n \rightarrow R^m$. Esta extensión permite evitar el desordenado y a veces virtualmente imposible procedimiento de insertar el dominio de f en el simplex standar n-dimensional. Así mismo, la técnica permite flexibilizar tamaños iniicales e intermedios de la retícula.

En su formulación de programación no-lineal[6] computando óptimos globales como locales según las funciones objetivo sean convexas o no, y según que las restricciones y las funciones objetivo sean continuamente diferenciales o no. En el segundo caso, como se ha afirmado, cualquier sub-gradiente puede sustituir a los gradientes.

4. Véase Ahijado (1977 b) donde se exponen dos algoritmos de primera y segunda generación respectivamente. Y para una descripción intuitiva de la forma de operar de una primera que epitomiza todos los demás Ahijado (1977 a) recesión de Scarf (1973).
5. La exposición de los dos algoritmos de esta sección sigue a Wilmuth (1973).
6. En el workspace 21FIXEDPOINT generado por Wilmuth (1975) recibe el encabezamiento ARGMIN, mientras que en su versión «punto fijo» recibe el de FIXK3, donde la diferencia más resaltable es la introducción de una función objetivo.

Eaves y Saigal (1972) extienden la triangularización original (conjunto primitivo o vector de las variables objetivo) de Eaves sobre $S^n \times (o, \infty)$, triangularizando primero $R^n \times o$ con una triangularización de Kuhn de algún tamaño inicial *d*. Después, sobre cada simplex de la triangularización de Kuhn se establece una copia a «escala» de la triangularización de Eaves sobre $S^n \times (o, \infty)$, de tal forma que se facilite el movimiento entre los cilindros (re-emplazamientos o pivotaciones) a todos los niveles por encima de la base. Todo ello se ilustra en la figura 1, para $n = 3$.

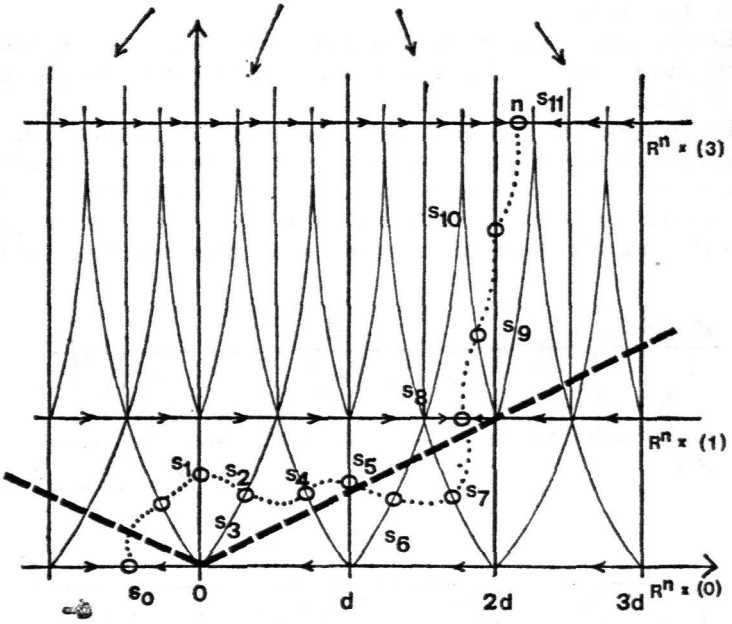

Figura 1. *Copias a escala de la triangularización de Eaves en $S^n \times \{o\}$*

Eaves y Saigal utilizan entonces un cono convexo, mostrado en la figura 1, como «vehículo» para establecer un simplex inicial, con el que comienza el algoritmo, y para conseguir que el «movimiento» obtenga refinamientos cada vez mayores de la retícula. Por debajo del cono, los vértices de la retícula se «etiquetan artificialmente» (siempre con vectores) para formar el movimiento hacia «adentro» del cono. Dentro del mismo, los vértices se etiquetan (también siempre con vectores) de tal forma que reflejan una acción real de la función sobre el vértice) proyectado sobre la base $R^n \times \{o\}$. Esta técnica provee de un simplex «de salida» en el vértice del cono. La etiquetación artificial mantiene la operación del

algoritmo dentro del cono. Dado que existe un número finito de simplices por debajo de cualquier nivel del cono, y dado que nunca se revisita un simplex, la sucesión $\{s_0, s_1, \ldots\}$ alcanzará ese nivel de finura de la retícula en un número finito de iteraciones.

El algoritmo de Merrill(o algoritmo de la clase $R^n \times (o, 1)$)

El algoritmo de Merrill es el único elemento de esta clase. Como en el algoritmo de Eaves-Saigal, maneja funciones sobre R^n y por tanto también evita el procedimiento de insertar el dominio de f en el simplex standard n-dimensional.

El algoritmo imprime una triangularización simple de una finura de retícula dada d, sobre la región $R^n \times (o, 1)$, como se muestra en la figura 2, para $n = 1$.

En un corte de nivel de $R^n \times \{1\}$, los vértices se etiquetan de modo que reflejan la acción de la función f *sobre* sus proyecciones en R^n. Los vértices interiores a $R^n \times \{o\}$, se etiquetan artificialmente de forma que nos provean de un simplex inicial s_o de $R^n \times \{o\}$, y de tal manera que contenga el punto inicial deseado tal que $x^o \in R^n$. Esta etiquetación artificial

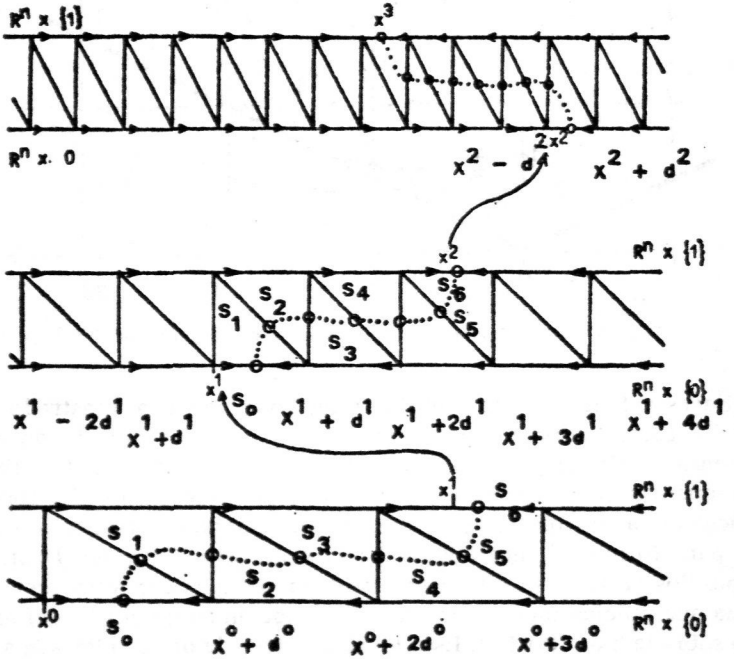

Figura 2. *Triangularizaciones de Merrill en $R^n \times 0,1$.*

también se elije de tal forma que garantice que el algoritmo no revisita ningún simplex n-dimensional de $R^n \times \{o\}$ distinto del iniical.

Bajo ciertas condiicones de f una sucesión finita $(s_o, s_1, \ldots s_t\}$ de simplices n-dimensionales se generará de tal manera que $s_t \subset R^n \times \{1\}$ y tal que la proyección de s_t sea un punto fijo aproximado de f.

Ahora puede refinarse la retícula, y «recomenzar» (son algoritmos de la familia del «restart») el algoritmo con $x^1 \in$ conv (s_t) como nuevo punto inicial. Los sucesivos refinamientos y recomienzos del algoritmo, producirán una sucesión de puntos fijos *aproximados* que convergerán a puntos fijos de f.

En este algoritmo el refinamiento de la retícula puede tener cualquier medida, e incluso ser diferente en cada estadio. Para alcanzar esta flexibilidad, en cada refinamiento se debe reintroducir la etiquetación artificial, posiblemente arrojando información útil, y por lo tanto incrementando la probabilidad de que se produzca un número extra de iteraciones. Sin embargo esta pérdida de «realización», puede compensarse por la sencillez, en una utilización ya de por si compleja.

III

Como ejemplos, se presentan los resultados ordenados de la computación de ocho problemas de test debidos a Colville, ya mencionados.

Para mantener la homogeneidad con las expresiones *APL.SV*, reescribiremos el problema genérico como

$$\begin{aligned} &\min \ OO\ (X) \\ &\text{S.T.} \quad GG \leq o \end{aligned}$$

Las características de los mismos vienen resumidas en el siguiente cuadro:

Problemas	*Dimensión*	*F*	*OO*	*GG*	*Variables Globales*	*Funciones Lamadas*
1	5	FC1	OOC1	GGC1	A1, B1, C1, D1, E1.	
2	15	FC2	OOC2	GGC2	A1, B1, C1, D1, E1.	
3	5	FC3	OOC3	GGC3	A3, B3, C3, D3, E3, M3.	
4	4	FC4	OOC4	GGC4	——	
5	6	FC4	OOC5	GGC5	——	
6	2	FC6	OOC6	GGC6	—	
7	8	FC7	OOC7	GGC7	A7, B7, C7,	
8	3	FC8	OOC8	GGC8	——	Y28

Donde la dimensión se refiere al número de variables de la función objetivo; las *FC* son los encabezamientos de los programas *APL* que describen cada uno de los problemas; *OO* y *GG* conservan los significados antedichos, es decir funciones objetivo y restricciones; las variables globales sirven para *asignar* los valores numéricos a los parámetros, y las funciones «llamadas» son funciones o programas de carácter auxiliar que sin ser algoritmos ni programas-descripción-de-problemas, partenecen al área de trabajo de los algoritmos, y les «ayudan» rebajando la carga computacional de los mismos.

Las variables globales vienen descritas por las siguientes matrices y vectores:

$$A1 = \begin{bmatrix} -16 & 2 & 0 & 1 & 0 \\ 0 & -2 & 0 & 0{,}4 & 2 \\ -3{,}5 & 0 & 2 & 0 & 0 \\ 0 & -2 & 0 & -4 & -1 \\ 0 & -9 & -2 & 1 & -2{,}8 \\ 2 & 0 & -4 & 0 & 0 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -2 & -3 & -2 & -1 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$A7 = \begin{bmatrix} 1 & 0 & 0 & 0{,}5 & 0 & 0 & 0{,}5 & 0{,}5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0{,}5 \\ 0 & 1 & 0{,}5 & 0 & 0 & 0 & 0{,}5 & 0 & 0 & 0{,}5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0{,}5 & 1 & 0 & 0 & 0 & 0{,}5 & 0 & 0{,}5 & 0{,}5 & 0 & 0 & 0{,}5 & 0{,}5 & 0 & 0 \\ 0{,}5 & 0 & 0 & 1 & 0 & 0 & 0{,}5 & 0 & 0 & 0 & 0{,}5 & 0 & 0 & 0 & 0{,}5 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0{,}5 & 0 & 0 & 0{,}5 & 0 & 0{,}5 & 0 & 0 & 0 & 0{,}5 & \\ 0 & 0 & 0 & 0 & 0{,}5 & 1 & 0 & 0{,}5 & 0 & 0 & 0 & 0 & 0 & 0{,}5 & 0 & 0 \end{bmatrix}$$

$$B7 = \begin{bmatrix} 0{,}22 & 0{,}2 & 0{,}19 & 0{,}25 & 0{,}15 & 0{,}11 & 0{,}12 & 0{,}13 \\ -1{,}45 & 0 & -1{,}3 & 1{,}82 & -1{,}15 & 0 & 0{,}8 & 0 \\ 1{,}29 & -0{,}89 & 0 & 0 & -1{,}16 & -0{,}96 & 0 & -0{,}49 \\ -1{,}1 & -1{,}6 & 0{,}95 & -0{,}54 & 0 & -1{,}78 & -0{,}41 & 0 \\ 0 & 0 & 0 & -1{,}43 & 1{,}51 & 0{,}59 & -0{,}33 & 0{,}43 \\ 0 & -1{,}72 & -0{,}33 & 0 & 1{,}26 & 1{,}24 & 0{,}21 & -0{,}26 \\ 0 & 0{,}45 & 0{,}26 & -1{,}1 & 0{,}58 & 0 & -1{,}3 & 0{,}1 \end{bmatrix}$$

$$C1 = \begin{bmatrix} 30 & -20 & -10 & 32 & -10 \\ -20 & 39 & -6 & -31 & 32 \\ -10 & -5 & 10 & -6 & -10 \\ 32 & -31 & -36 & 39 & -20 \\ -10 & 32 & -10 & -20 & 30 \end{bmatrix}$$

A3 = [5,3578547 0,8356891 37,293239 —40792,141]
B1 = [—40 —2 —0,25 —4 —4 —1 —40 —60 5 1]
B3 = [0 90 20]
C3 = [92 110 20]
C7 = [2,5 1,1 —3,1 —3,5 1,3 2,1 2,3 —1,5]
D1 = [4 8 10 5 2]
D3 = [78 33 27 27 27]
E1 = [—15 —27 —36 —18 —12]
E3 = [102 45 45 45 45]

$$M3 = \begin{bmatrix} 8{,}5334407E1 & 5{,}68580000E^{-}3 & 6{,}26200000E^{-}4 & 0 \\ 8{,}51249000E1 & 7{,}13170000E^{-}3 & 2{,}995500000E^{-}3 & 2{,}181300000E^{-}3 \\ 9{,}30096100E0 & 4{,}702600000E^{-}3 & 1{,}254700000E^{-}3 & 1{,}908850000E^{-}3 \end{bmatrix}$$

Los resultados computacionales

Los algoritmos de Eaves-Saigal, y Merrill son funciones monádicas en la terminología *APL*, que reciben como argumentos a la derecha del encabezamiento (en estos casos ARGMIN y ARMINNER respectivamente) los siguientes parámetros:

1) dimensión.
2) cota de error
3) grado inicial de finura de la retícula
4) un parámetro fijo, generalmente 0,5 (para su significado ver Wilmuth [1973])
5) estimación inicial.

En consecuencia una vez asignados los parámetros del problema, los programas APL que definen cada problema se ponen en relación con los algoritmos que los computan mediante una sencilla operación de «teclado» del tipo

x < — ARGMINMER 5 0 01 5 4 3 2 1 b 0,5 1 1 1 1 1

por ejemplo, para el algoritmo de Merrill y el tercer problema de Colville, y donde b es un meta-símbolo que significa «blanco» (espacio).

Los resultados obtenidos pueden resumirse[7] en

Colville 1 — Algoritmo Eaves-Saigal

	(primera cota 0,01)	(primera cota 0,1)
Iteraciones	117	160
Segundos	24,3833	32,5333
X	0,265729	0,265728
	0,321204	0,321204
	0,409645	0,409645
	0,439178	0,439178
	0,249394	0,939494
Segundos iteración	0,208974	0,20375

Colville 1 — Algoritmo de Merrill

Iteraciones	1387				
segundos	138,5				
finura retícula	0,0006	0,0004	0,0003	0,0002	0,0001
X	0,299638	0,333326	0,399886	0,428568	0,22059
GG (X)	—36,301	—1,952	0,0009	—1,395	—0,001 0,001
	—0,001	—38,314	—56,7529		
OO (X)	—32,3407				
Δ OBJ	0,0423669				
segundos iteración	0,099039				

Colville 2. — Algoritmos Eaves-Eaigal y Merrill

No converge.

Colville 3. — Algoritmo Eaves-Saigal

7. El terminal imprime resultados por bloques de iteraciones, por ejemplo, 731, 1086, 1140, 1420, 1424, 1428, 1432, 1436, 1440, 1444, 1448, 1452, 1456, para el Colville 6. Aquí solo recogeremos los resultados del último bloque dejando para las figuras reproducidas por el Plotter la evolución de la finura de la retícula y el núcleo de iteraciones. Véase más abajo.

Iteraciones	17.172				
segundos	3563,23				
finura					
retícula	0,195312	0,195312	0,195312	3,90625E-5	0,117187
X	76,1717	31,3619	30,3854	30,0754	30,1199
OO (X)	31091,1				
Δ OBJ	9,35228				

Colville 3. — Algoritmo de Merrill

Iteraciones	1260				
segundos	142,05				
finura					
retícula	2,28882E-5	4,57764E-5	3,05176E-5	1,525888E-5	3,8147E-5
X	78	33	29,9953	45	36,7758
GG (X)	—92	—8,84051	2,43298E-6	2,00856E-6	—11,1595E-5
OO (X)	—30665,5				
Δ OBJ	0,00568914				
segundos					
iteración	0,112778				

Colville 4. — Algoritmo Merrill

Iteraciones	196			
segundos	33,3333			
finura				
retícula	0,015625	0,016625	0,015625	0,15625
X	0,995703	0,991611	1,00404	1,00812
Δ OBJ	1,00187781			
segundos				
iteración	0,112782			

Colville 4. — Algoritmo de Eaves-Saigal

Iteración	159			
finura				
retícula	0,03125	0,03125	0,03125	0,03125
segundos	35,2			
X	0,88844	0,877817	1,00801	1,01682
segundos				
iteración	0,223312			

Colville 5. — Algoritmo de Eaves-Saigal

Iteraciones	1954
segundos	838,583
finura	
retícula	1,5 0,5 0,5 0,5 0,5
X	5,5002 3,49982 44,0066 7,49364 1
Δ OBJ	1E15

Colville 5. — Algoritmo de Merrill

Requiere un tiempo de CPU prohibitivo. (Véase nota respecto a Colville 7.)

Colville 6. — Algoritmo de Eaves-Saigal

Iteraciones	1753
segundos	246,15
finura	
retícula	0,00195312 0,000976562
X	20 0,0756978
GG (X)	—103,137 —42,8349 989,812 —201,301 —896,863
	—37,1651 —1010,19 —79,9998 —0,0756978 —0,000156
OO (X)	8851,93
Δ OBJ	0,0465762
segundos	
iteración	0,169488

Colville 6. — Algoritmo de Merrill

Iteraciones	1456
segundos	246,683
finura	
retícula	0,000488281 0,000244141
X	20 0,0757484
GG (X)	—103,199 —42,8302 —989,827 —201,363 —896,801
	—37,1698 —1010,17 —79,9989 —0,0757484 —0,0001164
	—0,447852
OO (X)	851,8
Δ OBJ	0,0994035
segundos	
iteración	0,169517

Colville 7. — Algoritmos de Eaves-Saigal y Merrill

Requieren un tiempo de Unidad Central de Procedimiento absolutamente prohibitiva.

Colville 8. — Algoritmo Eaves-Saigal

Iteraciones	515
segundos	263,35
finura	
retícula	0,375 0,25 0,125
X	2,54265 24,1577 0,172408
OO (X)	—1,64648
Δ OBJ	0,00879605
segundos	
iteración	0,511845

Colville 8. — Algoritmo de Merrill

Iteraciones	696
segundos	255,267
finura	
retícula	0,75 0,5 0,25
X	0,285182 —0,00246934 123,503
GG (X)	0,285182 0,00246834 —128,503 —1999,71 —16000
	8,50251 —0,335 —0,123518 —129986 0,266249
	2,57554 —5,57332 8,79875 —4999,66 —1999,88
	4,99857 —5,26625 —11,5755 1,58332 —25,7987
OO (X)	1284,98
Δ OBJ	0,0338313
segundos	
iteración	0,367098

8. Con una asignación del tipo X <--- ARGMINMER 8b0.01b8b7b6b5b4b3b2b1b0.5b1b1b1b 1b1b1b1b1b (la que es plausible además de similar a las restantes realizadas implementadas) después de 51 minutos de CPU se decidió «abortar» la realización considerando su prohibitividad. Hasta entonces había impreso 319 iteraciones sobre 7000 esperadas. Se produjo un «resultado» muy parecido con el ARGMIN (sólo que a quince minutos del comienzo).

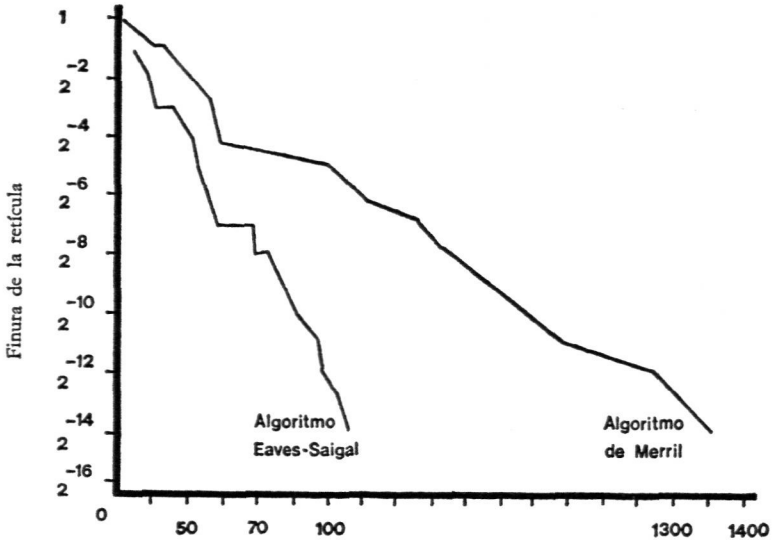

Figura 3. Colville 1

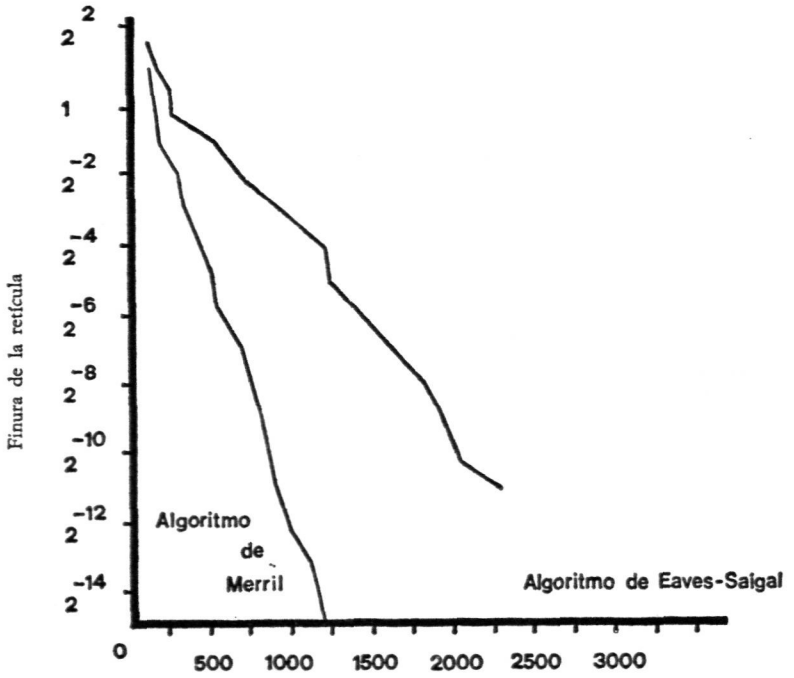

Figura 4. Colville 3

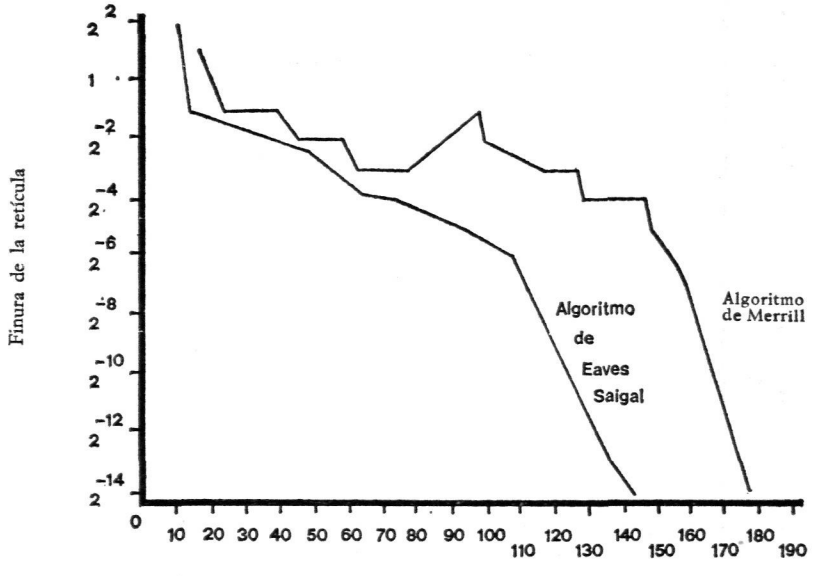

Figura 5. Colville 4.

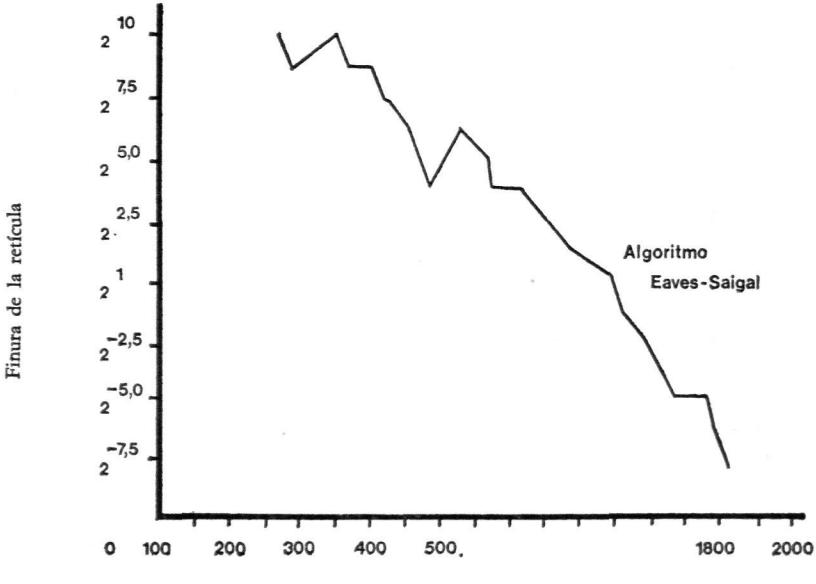

Figura 6. Colville 5.

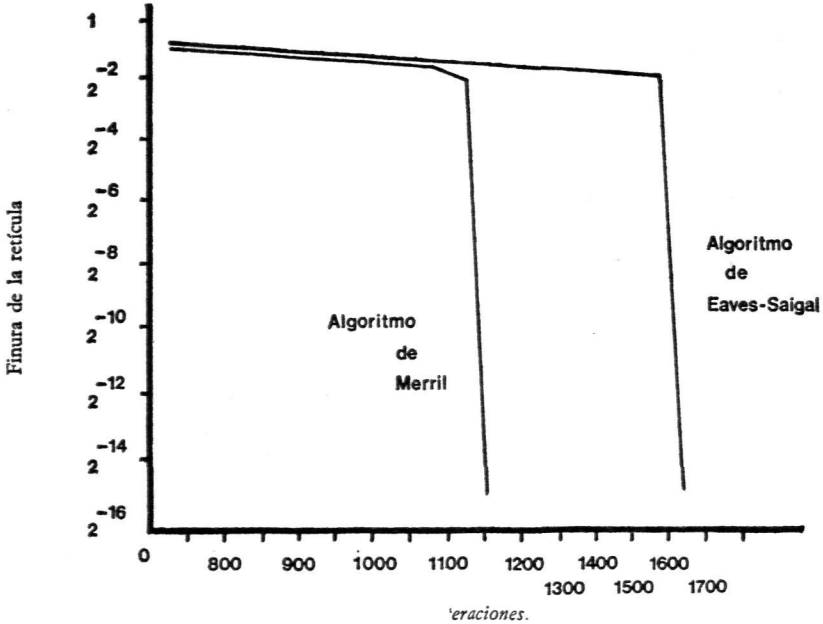

Figura 7. Colville 6

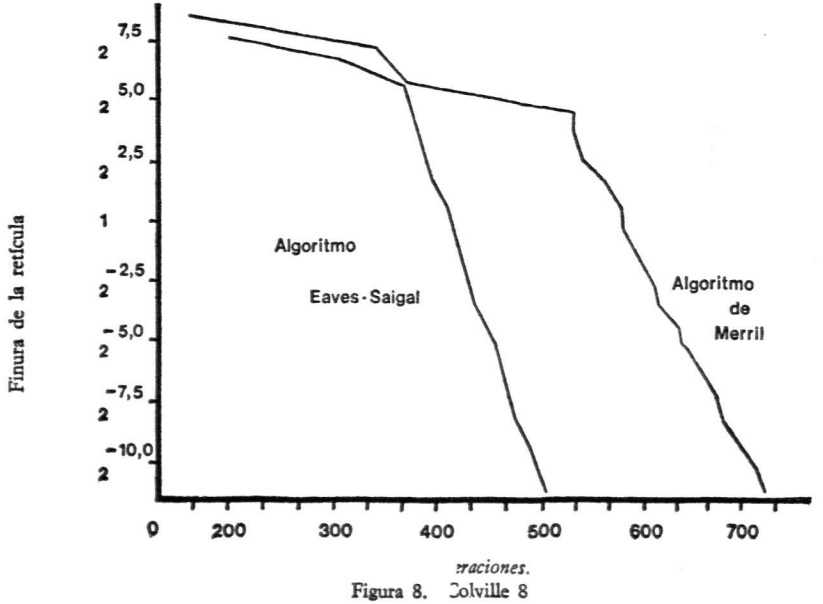

Figura 8. Colville 8

CUADRO RESUMEN

		Segundos por iteración
Colville 1.	Algoritmo Eaves-Saigal	0,208
	Algoritmo de Merrill	0,099
Colville 2.	Algoritmo Eaves-Saigal	No converge
	Algoritmo de Merrill	No converge
Colville 3.	Algoritmo Eaves-Saigal	0,047
	Algoritmo de Merrill	0,112
Colville 4.	Algoritmo Eaves-Saigal	0,223
	Algoritmo de Merrill	0,112
Colville 5.	Algoritmo Eaves-Saigal	0,110
	Algoritmo de Merrill	CPU Prohibitivo
Colville 6.	Algoritmo Eaves-Saigal	0,073
	Algoritmo de Merrill	0,169
Colville 7.	Algoritmo Eaves-Saigal	CPU Prohibitivo
	Algoritmo de Merrill	CPU Prohibitivo
Colville 8.	Algoritmo Eaves-Saigal	0,511
	Algoritmo de Merrill	0,367

Eficiencia Computacional de los algoritmos.

BIBLIOGRAFIA

AHIJADO, M.: «Computación de modelos de equilibrio general: Una primera aproximación». *Investigaciones Económicas*, n.º 4, septiembre-diciembre (1977 b).

COLVILLE, A. R.: «A comparative study of non-linear programming codes». IBM, New York Scientific Center Reort n.º 320-2949 (1968).

LEMKE, C. E.; HOBSON, J. T.: «Equilibrium points of bimatrix games». SIAM Journal of Applied Mathematics (1964).

MERRILL, O. H.: «Applications and extensions of an algorithm that computes fixed points of certain non-convex upper semicontinuous point to set mappings». Technical Report, Department of Industrial Ingineering, University of Michigan.

ROCKAFELLAR, R. T.: *Convex Analysis.* Princeton University Press.

KÜNZI, H. P.; KRELLE, W.: *Nichlineare Programmierung.* Springer Verlag. (1962).